INFORMATION AND COMPUTATION **128**, 57–71 (1996)
ARTICLE NO. 0063

Randomized Coordinated Attack*

GEORGE VARGHESE†

Washington University, St. Louis, Missouri 63130-4899

AND

NANCY A. LYNCH‡

Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

We study *randomized, synchronous* protocols for coordinated attack. Such protocols trade off the number of rounds (N), the worst-case probability of disagreement (U), and the probability that all generals attack ($\mathscr{L}$). We prove a nearly tight bound on the tradeoff between $\mathscr{L}$ and U ($\mathscr{L}/U \leqslant N$) for a *strong* adversary that destroys any subset of messages. Our techniques may be useful for other problems that allow a non-zero probability of disagreement

1. INTRODUCTION

Suppose two computers are trying to perform a database transaction over an unreliable telephone line. If the line goes dead at some crucial point, standard database protocols mark the transaction status as "uncertain" and wait until communication is restored to update its status. The protocol will ensure that the two computers *eventually* agree if communication is eventually restored.

On the other hand, suppose that the transaction has a real-time constraint (e.g., a decision to commit or reject the transaction must be reached in 10 min) and the cost of disagreement is high. Then standard commit protocols do not work. If communication can fail for up to ten minutes it is always possible for the two computers to disagree. Is there a protocol that prevents disagreement in all cases?

The answer is no. The question was first formalized in [8] as the coordinated attack problem. In this problem, there are two generals who communicate only using unreliable messengers. The generals are initially passive; however, at any instant either general may get an input signal that instructs him to try to attack a distant fort. The generals have a common clock. The problem is to synchronize attack attempts subject to the following conditions:

- *Validity*: If no input signal arrives, neither general attacks.[1]
- *Agreement*: Either both generals attack or both do not attack.
- *Nontriviality*: There is at least one execution of the protocol in which both generals attack.

Coordinated Attack and Network Protocols. It is possible to show that there is no deterministic solution to the coordinated attack problem (see related work section below). Note that link failures, and especially failure of all links, are a likely scenario in a real network. Thus the impossibility of coordinated attack (CA) fundamentally affects the way real network protocols are designed. For instance, when a transport (e.g., TCP) connection is disconnected, one end of the connection cannot get rid of its state after "being sure" that the other end has also done so. Instead, TCP has to rely on timers. Similar reasoning shows that when routes change in a datagram network (e.g., IP), one router must adopt the change before the others: this in turn can lead to inconsistent routes and packet looping. Information on TCP and IP, and other transport and routing protocols can be found in any network protocol text (e.g., [17]).

These two examples are instances of CA. CA essentially shows that it is impossible to ensure that two nodes can synchronize their state at the same time, in an environment where state changes can occur and links can fail for unbounded amounts of time. While our work on CA below is stated in a synchronous model for simplicity of notation,

* A preliminary version of this paper appeared in the 11th ACM Symposium on Principles of Distributed Computing, Aug 1992.

† Work done while at Laboratory for Computer Science, MIT.

‡ Supported by NSF Grant 8915206-CR, DARPA Grant N0014-89-J1988, and ONR Grant N0014-91-J1046.

[1] Another validity condition that is often used is that if no messages are delivered, then no general attacks. We prefer our definition because it focuses on input-output behavior. However, our results can be modified to fit the other validity condition.

0890-5401/96 $18.00

both the algorithm and lower bound we develop apply to an asynchronous (and hence practical) setting.

Related Work. It is shown in [8, 10] that there is no *deterministic* algorithm that meets all three conditions. In this paper, we consider a generalization to an arbitrary number of generals connected by a graph of unreliable links. Clearly, the impossibility result applies here as well.

Coordinated attack (CA) looks suspiciously like *Byzantine agreement* (BA) [13], sometimes referred to as *distributed consensus.* In the traditional definition of BA, *generals* exhibit arbitrary failures, while in CA only *links* fail by destroying messages; also, in BA, there is a bound on the number of failures, while in CA, *all* links can be faulty. There does not appear to be any way to reduce CA to BA or vice versa.

Later variants of the consensus problem considered network connectivity (e.g., [5, 9]) link failures and send-omission failures (e.g., [18]) in which some of the messages sent by a process can be lost. Clearly consensus with an arbitrary number of send omission faults would be identical to CA; as far as we know, the work on send omission faults also assumes a bound on the number of failures. The ability to lose *all* messages, however, is crucial in proving impossibility results for CA.

A large amount of work on randomized Byzantine Agreement has been reported; [3] contains a survey of this work. There are, however, only a few known lower bound results for randomized consensus [3]. All the lower bounds for randomized consensus (e.g., [11, 7]) apply only to the case of arbitrary process failures. For example, [11] shows that in randomized Byzantine Agreement if the number of malicious processes is more than a third of the number of processes and if no authentication is available, then there is an adversary that causes the protocol to fail with probability at least 1/3. The results and techniques used for proving lower bounds for randomized BA look very different from the results in this paper for randomized CA.

How do standard database commit protocols (e.g., two and three phase commit protocols [2]) deal with the problem of agreement over unreliable links? They do so by introducing another output state "Undecided"; it is legal for one process to output "Undecided" while another process decides "Attack," as long as the first process *eventually* decides to "Attack." However, if there is a fixed time to decide (for example, a pair of computers deciding on whether a rocket should be launched in 5 s), then the "Undecided" state is not useful. Thus, as we alluded to earlier, coordinated attack deals with the problem of reaching agreement in a fixed amount of time using a network of unreliable links. We model the fixed amount of time available for a decision by a fixed number of rounds N after which every process must decide either "attack" or "not attack."

Randomized Coordinated Attack. There is a well-known history of *randomization* providing a cure for a *deterministic* impossibility result (e.g., [16, 1]). Thus, we turn to *randomized CA*. We hope to trade a small probability of disagreement when links fail for a high probability of agreement (on a positive outcome) when links do not fail.

We modify the correctness conditions for deterministic CA to fit randomized CA. We retain the validity condition. We modify the agreement condition by requiring that the worst-case probability of disagreement (denoted by U for *unsafety*) be smaller than ε, a parameter. We replace the nontriviality condition by a measure $\mathscr{L}(R)$ (for liveness) that measures the probability all generals attack after an input signal, given that messages are delivered according to a given pattern R. We measure the goodness of a CA protocol by seeing how high $\mathscr{L}(R)$ can be for a given R and ε.

It may seem strange that unsafety is measured as the worstcase across all runs while liveness is measured separately for each run. However, the situation is analogous to data link protocols in which safety must always be preserved (i.e., the sequence of delivered messages must always be a prefix of the sequence of sent messages) but liveness (i.e., every sent message is delivered) is guaranteed only if the channel is delivering messages.

Coordinated attack captures the fundamental difficulty of bounded time synchronization over unreliable message channels. his paper investigates whether randomization can help coordinated attack. Our answer is basically no for a strong adversary, and a qualified yes for much weaker adversaries. Our paper concentrates on a *strong adversary* that can deliver messages according to any possible pattern R but has no access to message bits. Some form of data encryption could be used to make this assumption reasonable in an insecure environment.

The only previous work on randomized CA is for a model in which messages can be lost with a known constant probability (e.g., [18]). But the solution for this case is simple: simply send a FIRE message repeatedly if the loss probability is less than 0.5. One copy will reach the other end with high probability that improves exponentially with the number of rounds. But no protocol designer would rely on such a protocol in practice because *real links crash* (and can lose all messages till the time arrives for a decision to be made). Thus this paper asks, for the first time, the basic question: can randomization help solve a fundamental problem in synchronizing state under a realistic model in which links can crash and restart?

Our adversary is deterministic and non-adaptive and cannot read message bits. Thus, this adversary is not the strongest adversary one could use. However, our version of a strong adversary does model real life settings where links can crash and restart at an arbitrary frequency. Since our results are pessimistic, and apply to practical settings,

there does not seem to be any point in considering stronger adversaries.

Paper Organization. The rest of this paper is organized as follows. Section 2 contains our model. Section 3 describes a simple but inefficient protocol that is used to illustrate the issues involved. Section 4 describes the main idea behind a lower bound for the special case of two generals. Moving to the general case requires some concepts that are described in Section 5. In Section 6, we use these concepts to prove a basic lower bound for the general case. Section 7 describes an "optimal" protocol against a strong adversary; the structure of this protocol is based closely on the proof of the lower bound. A small gap between the performance of our optimal protocol and the lower bound leads to a second, more refined, lower bound that is described in Section 8. Section 9 contains our conclusions. Finally, the two appendixes contain proofs of the second lower bound and the invariants of the optimal protocol. While there are a number of details for completeness, the main ideas can be found in Sections 4, 6, and 7.

2. MODEL

The generals are represented by processes i that are at the vertices of an undirected graph $G(V, E)$ with $V = \{1, ..., m\}$, $m \geqslant 2$. We consider synchronous protocols that work in $N+1$ rounds, numbered 0, ..., N, $N \geqslant 1$. Let *time* r denote the point in time after round r. Thus, time 0 denotes the end of round 0. For convenience, we let -1 denote the time at which round 0 starts.

We model the input as a message sent by a fictitious "environment" node v_0, that is sent in round 0. We assume $v_0 \notin V$. Informally, if a process i receives a message in round 0 from v_0, the corresponding "general" has received a signal to try to attack. Each process i also has, as part of its state, a sequence of J random bits called α_i. J is an upper bound on the total number of random bits used by any general.

A *protocol* F consists of a number of *local protocols* F_i. Each F_i, $i \in V$, is a state machine executed by process i. F_i has two possible start states, s_i^0 and s_i^1, a state transition function, δ_i, and a message generation function, σ_i. Let S_i^r be the set of messages *received* by i from its neighbors in round r. Let q_i^r be the state of i at time r for all r, $0 \leqslant r \leqslant N$. Then $q_i^r = \delta_i(q_i^{r-1}, r, S_i^r, \alpha_i)$. We assume without loss of generality that each process sends a message to each neighbor in rounds 1, ..., N, since we can always simulate algorithms in which this is not true by sending null messages that are ignored by the receiver. Let m_{ij}^r be the message *sent* by i to neighbor j in round r. Then $m_{ij}^r = \sigma_i(q_i^{r-1}, j)$. At time N, i outputs a bit that is a function, O_i, of q_i^N. $O_i(q_i^N) = 1$ if and only if general i decides to attack.

An execution of F is a vector of local executions. A *local execution* E_i consists of q_i^0, (m_{ij}^r for all neighbors j of i, S_i^r, q_i^r) for $1 \leqslant r \leqslant N$. To generate an execution of F we need to define a run that represents the inputs as well as which messages get through in rounds 1, ..., N of the protocol. Formally, a *run* $R = I(R) \cup M(R)$. $I(R)$, the *input* for run R, is an arbitrary subset of $\{(v_0, i, 0): i \in V\}$. $M(R)$, the *messages delivered* in run R, is an arbitrary subset of $\{(i, j, r): (i, j) \in E, 1 \leqslant r \leqslant N\}$.

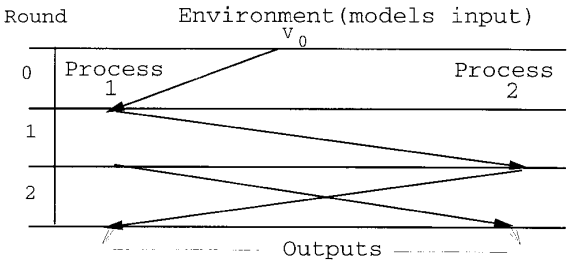

FIG. 1. Graphical depiction of the run $\{(v_0, 1, 0), (1, 2, 1), (1, 2, 2), (2, 1, 2)\}$.

For example, with two processes and two rounds, in the run $\{(v_0, 1, 0), (1, 2, 1), (1, 2, 2), (2, 1, 2)\}$ only process 1 receives a signal to attack. Also, in round 1, only the message sent from process 1 to process 2 is delivered; the message sent from process 2 to process 1 is lost. In round 2, messages sent by both processes are delivered. Figure 1 shows a graphical depiction of this simple run, where only delivered messages are sketched. The absence of a directed edge from i to j in round k implies that the corresponding message is lost.

We will use the notation (A_i) to denote a vector A consisting of a component A_i for each $i \in V$. An execution for a fixed F is uniquely determined by random input $\alpha = (\alpha_i)$, and a run R. We define $Ex(R, \alpha) = (E_i)$ as the *execution generated* by R and α for a fixed protocol F. Each E_i is a local execution such that:

- If $(v_0, i, 0) \notin R$ then $q_i^0 = s_i^0$. If $(v_0, i, 0) \in R$ then $q_i^0 = s_i^1$. (The initial state of the local execution encodes the input.)
- For all r, $1 \leqslant r \leqslant N$, and for all neighbors j of i: $m_{ij}^r = \sigma_i(q_i^{r-1}, j)$.
- For all r, $1 \leqslant r \leqslant N$, and for all neighbors j of i: $m_{ji}^r \in S_i^r$ iff $(j, i, r) \in R$.
- For all r, $1 \leqslant r \leqslant N$: $q_i^r = \delta_i(q_i^{r-1}, r, S_i^r, \alpha_i)$.

The *output* of execution E is the vector $(O_i(q_i^N))$. We say two executions E and $\tilde{E}$ are *identical* to j if $E_j = \tilde{E}_j$.

We consider sets of executions of a particular protocol. D_i denotes the set of executions in which $O_i(q_i^N) = 1$, and $\overline{D}_i$ the set of executions in which $O_i(q_i^N) = 0$. Let (D_i, R) denote the set of executions that have run R and in which $O_i(q_i^N) = 1$.

TA (*total attack*) denotes the set of executions $D_1 \cap D_2 ... \cap D_m$. *NA* (*no attack*) denotes the set of executions

$\overline{D_1} \cap \overline{D_2} \ldots \cap \overline{D_m}$. PA (*partial attack*) denotes the complement of $NA \cup TA$. Thus, TA is the set of executions in which all processes agree on an output of 1, NA is the set of executions in which all processes agree on an output of 0, and PA is the set of executions in which some pair of processes disagree.

Each α_i is drawn from $\{0, 1\}^J$ using the uniform probability distribution. This probability distribution on inputs α induces a probability distribution on executions for each possible run R, in the natural way. For each set X of executions and each run R, we use the notation $Pr[X, R]$ to denote the probability of event X according to this distribution of executions.

Now consider two runs $R = \{(i, j, 1)\}$ and $\tilde{R} = \varnothing$. The only difference in the runs is that the message from i to j is delivered in R. Thus, given the same random input, i will decide the same regardless of whether an execution follows run R or run $\tilde{R}$. This leads to a key notion of indistinguishable runs. We say that two runs R and $\tilde{R}$ are *indistinguishable* to i if for all α, $Ex(R, \alpha)$ and $Ex(\tilde{R}, \alpha)$ are identical to i. We use $R \stackrel{i}{\equiv} \tilde{R}$ to denote that R and $\tilde{R}$ are indistinguishable to i. A natural consequence is:

LEMMA 2.1 (Indistinguishability). *If* $R \stackrel{i}{\equiv} \tilde{R}$ *then* $Pr[D_i, R] = Pr[D_i, \tilde{R}]$.

Proof. Easy consequence of definitions.

An *adversary* $\mathscr{A}$ is a set of runs. We will only deal in this paper with the *strong adversary*, $\mathscr{A}_s$, where $\mathscr{A}_s$ is the set of all possible runs. (It is easy to conceive of stronger adversaries that can read message bits and use randomization. However, as we have argued in the introduction, the results obtained using our adversary will be sufficiently pessimistic.)

We define the *unsafety* of protocol F against strong adversary $\mathscr{A}_s$, as: $U_s(F) = Max_{R \in \mathscr{A}_s} Pr[PA, R]$. (Intuitively, this is the worst-case probability of partial attack on any run that can be produced by the adversary.)

Next, we describe the correctness conditions and the liveness measure. Validity requires that no general attacks if there is no input. Agreement requires that the worst-case probability of partial attack be no more than ε, a parameter. Finally, the liveness measure for a run R is the probability of total attack on run R.

- *Validity*: For all vectors α, for all R such that $I(R) = \varnothing$, and for all i, the output of all executions in $Ex(R, \alpha)$ is the vector consisting of all zeroes.
- *Agreement*: F satisfies agreement with parameter ε if $U_s(F) \leqslant \varepsilon$.
- *Liveness*: The *liveness* $\mathscr{L}(F, R)$ of protocol F on run R is $\mathscr{L}(F, R) = Pr[TA, R]$.

Our goal is to find an "optimal" algorithm F (that meets the validity condition, and the Agreement condition for a fixed ε) such that $\mathscr{L}(F, R)$ is as large as possible for any run R. We end this section with two elementary lemmas on which our lower bounds are based. The first states that the unsafety of a protocol in any run is at least as large as the difference in attack probabilities of any two processes. The second states that the liveness of a protocol is no more than the attack probability of any process. The two inequalities given below do not seem very tight, and so it is perhaps surprising that the lower bounds based on these inequalities are as tight as they are.

LEMMA 2.2 (Difference Bound). *For all* $i, j \in V$, $Pr[D_i, R] - Pr[D_j, R] \leqslant U_s(F)$.

Proof. Fix i and j. Using the definition of partial attack and the fact that the probability of the union of several events is greater than the individual probabilities,

$$Pr[PA, R] \geqslant Pr[D_i \cap \overline{D_j}, R].$$

The lemma follows because using the inclusion–exclusion formula we can easily show that $Pr[D_i \cap \overline{D_j}, R] \geqslant Pr[D_i, R] - Pr[D_j, R]$. ∎

LEMMA 2.3 (Liveness Bound). *For all* $i \in V$, $\mathscr{L}(F, R) \leqslant Pr[D_i, R]$.

Proof. Obvious from definition of liveness.

3. NEEDLE IN A HAYSTACK PROTOCOL

We informally describe a simple protocol A (which we call "the needle in a haystack" protocol for reasons that will become clear below) for two processes 1 and 2 against a strong adversary. The limitations of this protocol will motivate both the lower bound in Section 6 and the optimal protocol of Section 7.

Recall that we have assumed that each process must send some message (at least a null message) in every round. For convenience, let us call a non-null message (i.e., a message that carries information) a *packet*. Thus, we assume implicitly that on every round a process sends either a packet or a null message.

In rounds 1 through N, the two processes send packets to each other in alternate rounds (see Fig. 2). Process 2 is allowed to send packets in odd rounds, while process 1 is allowed to send packets in even rounds. The protocol begins with process 2 sending a packet in round 1 (see Fig. 2). Process 2 includes a bit b in its packet; b is set to 1 if and only if process 2 received a signal to attack in round 0. Process 1 sends a packet in round 2 if and only if it receives a packet in round 1 from process 2 *and* either $b = 1$ or process 1 received a signal to attack in round 0. In other words, process 1 will send a packet in round 2 if and only if it "knows" that one of the two processes has received an input signal to attack.

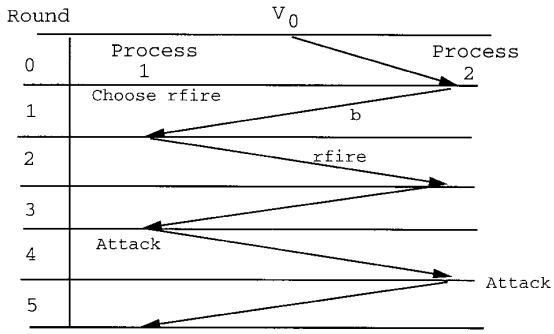

FIG. 2. Needle in a haystack protocol: sample run in which both processes attack. The protocol begins with process 2 sending a message in round 1 with a bit b that encodes process 2's input. Process 1 chooses a random number *rfire* (equal to 4) and sends it to process 2. Process 1 goes into the "attack" state at time $rfire - 1 = 3$ and process 2 goes into the "attack" state at time *rfire*. Effectively, a random "attack" message is hidden in a haystack of messages.

Before process 1 sends a packet in round 2, process 1 chooses a random integer *rfire* that is uniformly distributed in the range $[3, ..., N]$. Any packets sent by Process 1 contain only the value *rfire*; any packets sent by process 2 contain only the input bit b corresponding to process 2's input. If process 2 receives any packet from process 1, process 2 stores the value of *rfire*. In all rounds $r > 2$, a process i sends a packet in round r only if it has received a packet in the previous round, and $(r+1) \bmod 2 = i \bmod 2$. Thus, if the adversary destroys a packet sent in round r, all packet sending stops in rounds greater than r.

The decision rules are as follows. The default state of both processes is "no attack." However, if all packets sent in rounds 1, ..., $rfire - 1$, have been delivered, then the process that received the packet sent in round $rfire - 1$ (say process i) will transition to an "attack" state at time $rfire - 1$. If the next packet sent by process i in round *rfire* is also delivered, then the other process (say j) will also transition to an "attack" state at time *rfire*. Once a process is in the "attack" state, it stays in this state till the end of the execution. At the end of the execution, process i outputs "attack" iff process i is in the "attack" state at the end of the execution. Thus, in Figure 2 with $rfire = 4$, process 1 goes into the "attack" state at time 3, and process 2 goes into the "attack" state at time 4. On the other hand, if any packet sent before round *rfire* is destroyed, then both processes stop sending packets and output "no attack" at the end of the execution.

It is easy to see that this protocol satisfies validity. If neither process receives an input signal in round 1, then process 1 does not send a packet in round 2, and the protocol stops and both processes output "no attack."

Now consider unsafety. Since the adversary that controls message delivery does not know the value of *rfire*, the adversary has only a chance of approximately $1/N$ of causing partial attack. This is because the adversary can cause partial attack *only if the first packet destroyed in the run is the packet sent in round rfire*. Thus, $U_s(A) \approx 1/N$. In effect, we are placing a randomly chosen "needle" in a haystack of messages, and the adversary has only an inverse linear chance of guessing where the needle will lie.

Finally, let R_g be a "good" run in which all messages are delivered and the input is valid. Then on run R_g, both processes will always decide to attack. Hence $\mathscr{L}(A, R_g)$, the liveness of A on run R_g, is 1. However, this simple protocol raises two questions:

- $U_s(A) \approx 1/N$ and $\mathscr{L}(A, R_g) = 1$. Can we decrease $U_s(A)$ further while keeping $\mathscr{L}(A, R_g)$ unchanged? In other words, can we find a protocol (a) whose probability of making a mistake is better than $1/N$, and (b) whose probability of attacking on a good run is 1. It might seem that this can be done by running A several times. However, the answer is no, as we show in Section 6.
- Consider a run R in which an input signal to attack has been received, and all messages are delivered except the message sent by process 1 in round 2. It is easy to see that $\mathscr{L}(A, R) = 0$. Intuitively, this is not satisfactory because in run R, all but one message is delivered, and yet the probability of attacking on run R is 0. Can we design a protocol whose liveness grows in some fashion with the number of messages delivered in a run? We will describe an optimal protocol S in Section 7.

4. LOWER BOUND FOR TWO PROCESSES

We assume a fixed bound on unsafety, ε. We informally sketch the main idea behind a bound on liveness for the special case of only two processes. We will show that the probability of attack of any process i in run R is bounded by $\varepsilon L_i(R)$, where $L_i(R)$ is a measure that depends on the number of messages that get delivered in run R. Since we can show that $L_i(R) \leqslant N$,[2] we can show that the liveness of any two process protocol cannot exceed εN. This section is meant to give the reader the intuition behind the lower bound. In the next two sections, we will give a formal proof of the bound for the general case.

In Fig. 3, consider a typical run R (shown in the upper half of the figure) with two processes, processes 1 and 2. To bound the attack probabilities in run R, we consider the sequence of three runs $R_0, ..., R_2$ shown in the figure. Starting with the null run R_0 in which no message or input is received, we gradually increase the number of delivered messages so that the last run R_2 differs from the original run R only in the delivery of a single message.

Recall that the Indistinguishability Lemma (Lemma 2.1) states that a process will have the same attack probabilities

[2] Recall that N is the number of rounds.

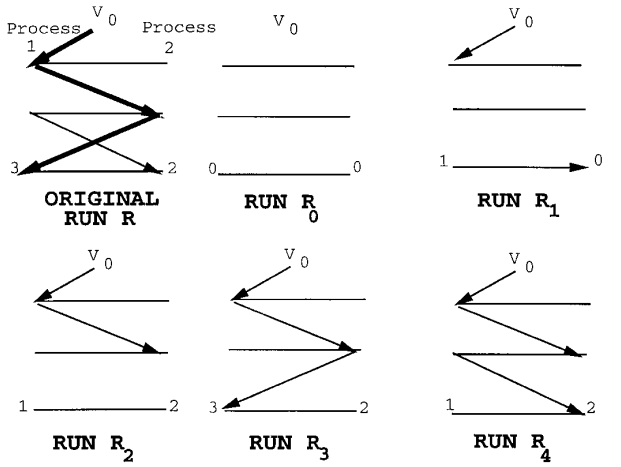

FIG. 3. Bounding the attack probabilities in a run R using a sequence of runs R_0, ..., R_2 and using the difference bound lemma. The bold lines in the original run form an *alternating chain*. The numbers at the end of a run give the length (L_i) of the longest alternating chain that ends at each process i; the attack probability for process i is bounded by εL_i.

on two indistinguishable runs. We will make heavy use of this lemma (without explicit reference) in what follows.

By the validity condition, in run R_0 both processes must have zero attack probabilities. But runs R_1 and R_0 are indistinguishable to process 2 and hence (by the Indistinguishability Lemma), process 2 has zero attack probability in R_1. But, by the Difference Bound Lemma, this means that $Pr[D_1, R_1] \leqslant \varepsilon$. Next, runs R_2 and R_1 are indistinguishable to process 1 and hence $Pr[D_1, R_2] \leqslant \varepsilon$. Hence, by the Difference Bound lemma, $Pr[D_2, R_2] \leqslant 2\varepsilon$. The same argument shows that $Pr[D_2, R_3] \leqslant 2\varepsilon$ and $Pr[D_1, R_3] \leqslant 3\varepsilon$.

Similarly, runs R_4 and R_1 are indistinguishable to process 1 and hence $Pr[D_1, R_4] \leqslant \varepsilon$ and $Pr[D_2, R_3] \leqslant 2\varepsilon$. Finally, returning to the original run R, we see that runs R and R_3 are indistinguishable to process 1, and runs R and R_4 are indistinguishable to process 2. Hence, $Pr[D_1, R] \leqslant 3\varepsilon$ and $Pr[D_2, R] \leqslant 2\varepsilon$. Hence, by the Liveness Bound Lemma, the liveness on run R is no more than 2ε.

In Fig. 3, in the sketch of run R we show in bold what we call an *alternating chain* that begins at V_0 and alternates between the two processes using delivered messages in the run. If we define $L_i(R)$ to be the length of the longest alternating chain that ends at process i in Run R, then it is not hard to show (using, say, induction on the round number) that $Pr[D_i, R] \leqslant \varepsilon L_i(R)$.

When we consider more than two processes, the concept of an alternating chain generalizes to that of *an information level*. Also, the proof of the general case is handled very simply using a tool called *causal clipping* (instead of using brute-force induction on round numbers). We define these concepts in the next section, and then prove the general lower bound formally in the following section.

5. INFORMATION FLOW, CLIPPING, AND INFORMATION LEVEL

In this section, we describe three concepts that underlie both the lower bounds of Section 6 and the protocol in Section 7. We begin with a definition that captures the idea of information flow or possible causality [12] between process-time pairs in a run.

Consider any $i, k \in V \cup \{v_0\}$ and any $r, s \in \{-1, 0, ..., N\}$. We say that (i, r) *directly flows to* (k, s) in run R if and only if $s = r + 1$ and either $i = k$ or $(i, k, s) \in R$. We define the *flows to* relation between process-time pairs as the reflexive transitive closure of the directly flows-to relation. Thus:

LEMMA 5.1 (Transitivity). *If* (i, r) *flows to* (j, s) *and* (j, s) *flows to* (k, t) *in run* R, *then* (i, r) *flows to* (k, t) *in run* R.

We introduce a measure of the "knowledge" [4, 10] a process has in a run. We first define information "height" and then use it to define the more useful idea of information "level."

The intuition behind the height definition is as follows. We say that a process "hears" about some event if there is information flow from the event to the process (see formal definition below.) Intuitively, a process reaches height 1 when it hears the input; a process reaches height $h > 1$ when it has heard that *all* other processes have reached height $h - 1$.

Formally, we say that j *can reach height* h by round r in run R if and only if h is a nonnegative integer subject to the following conditions:

- If $h = 0$, there are no conditions.
- If $h = 1$, $(v_0, -1)$ flows to (j, r) in R.

If $h > 1$, then for all $i \neq j \in V$, there is some r_i such that (i, r_i) flows to (j, r) in R and i can reach height $h - 1$ by round r_i in R.

Next, we define $L_j^r(R)$, the *level* j reaches by round r of run R, to be the *maximum* height j can reach by round r. We use $L_j(R)$ to denote $L_j^N(R)$ and $L(R)$ to denote $Min_{j \in V}(L_j(R))$. (Recall that N is the maximum round number.) Note that for the special case of two processes, $L_j(R)$ is equal to the length of the longest alternating chain in run R that ends at process j. Thus, the definition we gave in Section 4 is a special case of the general definition.

Finally, we introduce a construction to "clip" a run with respect to a process i such that the constructed run preserves all information flow to i. This construction is the key to the lower bound proof. We define $Clip_i(R) = \{(j, k, r) \in R: (k, r) \text{ flows to } (i, N)\}$ in run R. Figure 4 shows a sample run with three processes and two rounds. Assume the completely connected graph on the three nodes. After the causal clipping of this run with respect to process 1, all the messages shown in bold are removed from the run.

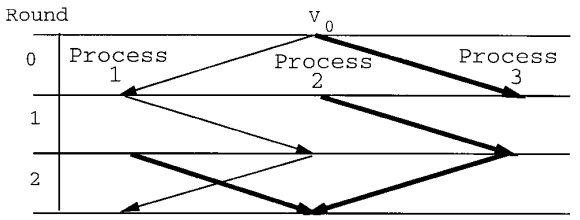

FIG. 4. Causal clipping with respect to process 1 removes all messages that do not causally flow to process 1 in the run. Such messages are shown in bold.

In essence, causal clipping with respect to process i removes all messages that do not contribute to i's knowledge in this run. Thus, it is not hard to see that clipping with respect to i preserves any information that i can gather in the run. Hence we have:

LEMMA 5.2 (Clipping). *Let* $Clip_i(R) = \tilde{R}$. *Then*:

- $L_i(R) = L_i(\tilde{R})$ *and*
- $R \stackrel{i}{\equiv} \tilde{R}$.

Proof. Straightforward consequence of model.

6. LOWER BOUND FOR STRONG ADVERSARY

The main idea behind the lower bound is to show that the probability of any process i attacking after any run R is no greater than the information level of Process i multiplied by the unsafety of the protocol—i.e., $Pr[D_i, R] \leqslant U_s(F)\, L_i(R)$. This directly leads to a bound on liveness of the protocol by the Liveness Bound Lemma, Lemma 2.3, since the liveness of the protocol on run R is no greater than the attack probability of any process.

Showing that $Pr[D_i, R] \leqslant U_s(F)\, L_i(R)$ would be easy if we could show that in run R there is some other process j with information level $L_i(R) - 1$. This follows from induction and the fact that the difference between the attack probabilities of any two processes cannot exceed the unsafety of the protocol (see the Difference Bound Lemma, Lemma 2.2). Unfortunately, it is easy to find runs in which all processes have the same information level.

Instead, we show that there is a clipped run $\tilde{R}$ that is indistinguishable to process i, and such that in $\tilde{R}$ there is indeed a "laggard" process j whose information level is strictly smaller than $L_i(R)$. Now we can apply the arguments described in the previous paragraph to show that $Pr[D_i, R] \leqslant U_s(F)\, L_i(R)$.

We build up to the final proof by a sequence of lemmas. The first lemma captures the intuitive idea that a process can change its level only by receiving a message.

LEMMA 6.1. *For any run* R *and any* $k \in V$, *if* $L_k(R) = l > 0$ *then there must be some tuple* $(j, k, r) \in R$ *such that* $L_k^r(R) = l$.

Proof. From the definition of level, we see that if there are no j, s such that $(j, k, s) \in R$ then $L_k^{s-1}(R) = L_k^s(R)$. Thus, if $L_k^N(R) = l$ we can work backwards from round number N until we find the r required for the lemma. If we fail then there is no $(*, k, *)$ tuple in R, which would imply that $l = 0$, a contradiction. Thus, we cannot fail. ∎

The next lemma describes the key property of clipped runs and information levels that we use to prove our lower bound. It says that if i reaches information level l at the end of run R then at the end of $Clip_i(R)$ there must be some process k (which we can call a *laggard*) whose information level lags behind that of process i. In essence, this is why i cannot go to a higher information level than l by the end of R.

LEMMA 6.2 (Laggard). *Consider a run* R *such that* $L_i(R) = l > 0$ *and* $Clip_i(R) = \tilde{R}$. *Then there is some* $k \in V$ *such that* $L_k(\tilde{R}) \leqslant l - 1$.

Proof. By contradiction. Thus, for all $k \in V$, we assume that $L_k(\tilde{R}) \geqslant l$.

Consider any $k \neq i$. By Lemma 6.1 and the fact that $l > 0$, there must be some tuple $(j, k, r) \in \tilde{R}$ such that $L_k^r(\tilde{R}) \geqslant l$. Since $(j, k, r) \in \tilde{R}$ then (by definition of clipping) (k, r) flows to (i, N) in R. Hence, we can show that (k, r) flows to (i, N) in $\tilde{R}$. We also know that $L_k^r(\tilde{R}) \geqslant l$. Since this is true for all $k \neq i$ we must have (see the definition of level) $L_i(\tilde{R}) \geqslant l + 1$. But by Lemma 5.2, this implies that $L_i(R) \geqslant l + 1$, a contradiction. ∎

We now formalize the result we alluded to in Section 4, that the attack probabilities can be bounded using the information level:

LEMMA 6.3. *For all protocols* F, *all runs* R, *and any process index* $i \in V$, $Pr[D_i, R] \leqslant U_s(F)\, L_i(R)$.

Proof. By induction on l in the following inductive hypothesis.

Inductive Hypothesis. For all i and all runs R with $L_i(R) = l$, $Pr[D_i, R] \leqslant U_s(F)\, l$.

Base case, $l = 0$: Thus, $L_i(R) = 0$. Let $\tilde{R} = Clip_i(R)$. We first claim that $I(\tilde{R}) = \{\ \}$. Suppose not, for contradiction. Then there is some j such that $(v_0, j, 0) \in \tilde{R}$; hence, since $\tilde{R} \subseteq R$, $(v_0, j, 0) \in R$. Also from the definition of clipping, $(j, 0)$ flows to (i, N) in R. But in that case, $L_i(R) \geqslant 1$, a contradiction. Thus, we must have $I(\tilde{R}) = \{\ \}$. Also by the Clipping Lemma (Lemma 5.2), $R \stackrel{i}{\equiv} \tilde{R}$. Hence $Pr[D_i, R] = Pr[D_i, \tilde{R}] = 0$, by the Indistinguishability Lemma (Lemma 2.1) and the validity requirement. Thus, $Pr[D_i, R] = U_s(F)\, l$.

Inductive step, $l > 0$: Consider any l and R such that $L_i(R) = l$. Let $\tilde{R} = Clip_i(R)$. By the Laggard Lemma (Lemma 6.2), there exists some k such that $L_k(\tilde{R}) \leqslant L_i(R) - 1$. Hence, by the inductive hypothesis, $Pr[D_k, \tilde{R}]$

$\leqslant U_s(F)(l-1)$. But by our bound on unsafety, Lemma 2.2, $Pr[D_i, \tilde{R}] - Pr[D_k, \tilde{R}] \leqslant U_s(F)$. Hence $Pr[D_i, \tilde{R}] \leqslant U_s(F)\, l$. But by the fact that R and $\tilde{R}$ are indistinguishable to i and by the Indistinguishability Lemma (Lemma 2.1), it follows that $Pr[D_i, R] \leqslant U_s(F)\, l$. ∎

The last lemma immediately leads to:

THEOREM 6.4 (Basic Lower Bound). *For any F, $\mathscr{L}(F, R) \leqslant U_s(F)\, L(R) \leqslant \varepsilon L(R)$.*

Proof. From Lemma 6.3, for any $i \in V$, $Pr[D_i, R] \leqslant U_s(F)\, L_i(R)$. Thus, from Lemma 2.3, $\mathscr{L}(F, R) \leqslant U_s(F)\, L_i(R)$ for any $i \in V$. Thus, from the definition of $L(R)$, $\mathscr{L}(F, R) \leqslant U_s(F)\, L(R)$. The theorem now follows from the Agreement condition. ∎

7. OPTIMAL PROTOCOL AGAINST A STRONG ADVERSARY

Our optimal protocol, Protocol S, is based closely on the lower bound arguments. Its structure is quite different from the simple protocol in Section 3 which was introduced to motivate the problem.

In Protocol S we will arbitrarily designate process 1 to choose a random number *rfire*. In order to output "attack," we will require that any other process i hear the value of *rfire* from process 1 in addition to hearing the input. This motivates a second measure on a run R that we call the *modified level* measure. We first describe this new measure and then give an overview of how the protocol calculates and uses this new measure. Finally, we give a formal description and proof.

7.1. *Modified Level Measure*

The modified level measure is defined in a parallel fashion to the original level measure by first defining a modified height or m-height. Formally, we say that process j *can reach m-height h* by round r in run R if and only if h is a non-negative integer subject to the following conditions:

- If $h = 0$, there are no conditions.
- If $h = 1$, $(v_0, -1)$ and $(1, 0)$ flow to (j, r) in R.
- If $h > 1$, then for all $i \neq j \in V$, there is some r_i such that (i, r_i) flows to (j, r) in R and i can reach m-height $h-1$ by round r_i in R.

Thus, the only difference between the m-height and height definitions is in the condition required to reach m-height 1. In the case of m-height we not only require that j has heard the input but also that j has heard from process 1. We also define $ML_i^r(R)$, $ML_i(R)$, $ML(R)$ analogously to the previous definitions for L_i.

Because of the small difference in the definitions, it is easy to show that the modified level measure differs by at most one from the level measure.

LEMMA 7.1. *For all R and $i \in V$, $L_i(R) - 1 \leqslant ML_i(R) \leqslant L_i(R) \leqslant N$.*

Proof. It is clear from the definitions that $ML_i(R) \leqslant L_i(R) \leqslant N$. We prove that $L_i(R) - 1 \leqslant ML_i(R)$ by induction on l in the following inductive hypothesis: for all R and $i \in V$, if $L_i^r(R) = l$ then $ML_i^r(R) \geqslant l-1$. The definition of $L_i^r(R) = l$ has a different form for the case when $l = 1$ and the case when $l > 1$. Thus, we use $l = 2$ as our base case. We note, however, that the case of $l = 1$ is trivial because it is easy to see from the definition that $ML_i^r(R) \geqslant 0$.

Base case, $l = 2$: From the definition of $L_i^r(R)$, there must some $s \geqslant 0$ such that $(1, s)$ flows to (i, r) and $L_1^s(R) = 1$. But if $L_1^s(R) = 1$, then by definition, $(v_0, -1)$ flows to $(1, s)$; hence by Transitivity (Lemma 5.1), $(v_0, -1)$ flows to (i, r). Also since $(1, 0)$ flows to $(1, s)$, by Transitivity (Lemma 5.1), $(1, 0)$ flows to (i, r). Hence, by definition, $ML_i^r(R) \geqslant 1$.

Inductive Step, $l > 2$: From the definition of $L_i^r(R)$, for every $j \neq i$ there must be some r_j such that (j, r_j) flows to (i, r) in R and $L_j^{r_j}(R) = l-1$. By the inductive hypothesis, $ML_j^{r_j}(R) \geqslant l-2$. Since this is true for all $j \neq i$, by using the definition of $ML_i^r(R)$, we have $ML_i^r(R) \geqslant l-1$. ∎

The second useful property is that the the modified level measured by any two processes can differ by at most one.

LEMMA 7.2. *For all R and $i, j \in V$, $ML_j(R) \geqslant ML_i(R) - 1$.*

Proof. Consider any $i, j \in V$. If $ML_i^r(R) = 1$ we are done trivially. Suppose $ML_i^r(R) \geqslant 1$. Then we know from the definition that there must be some $s < r$ such that (j, s) flows to (i, r) and $ML_j^s(R) = ML_i^r(R) - 1$. The lemma follows, because it is clear from the definition that $ML_j^r(R) \geqslant ML_j^s(R)$. ∎

7.2. *Protocol Overview*

We will design a protocol based closely on the lower bound arguments of the previous section. Recall that we had shown in our basic lower bound (Theorem 6.4) that for any F, $\mathscr{L}(F, R) \leqslant \varepsilon L(R)$. We have also seen that the modified level measure differs by at most one from the level measure. Thus, in order to come close to meeting the lower bound, we will design a protocol in which:

- Each process i will calculate $ML_i(R)$, the value of the modified level at the end of the current run R.
- Each process will output "attack" with a probability proportional to $ML_i(R)$. This causes the liveness of the protocol to grow with $ML_i(R)$.

To do so each process i in protocol S has a variable $count_i$ that counts the value of $ML_i^r(R)$. We say that i has begun counting if $count_i > 0$. We will see how i begins counting below. However, once i has begun counting, process i increases $count_i$ to s (for $s > 1$) when it has heard that all other processes have reached a count of $s-1$. It is easy to implement this if each message sent by a node i carries $count_i$ and a variable called $seen_i$, the set of nodes that i knows has reached $count_i$.

Protocol S must satisfy Agreement with parameter ε. Process 1 chooses a random number *rfire* uniformly distributed in the range $(0, 1/\varepsilon]$ and passes it on all messages. As we will see, it is important that *rfire* be a real (as opposed to the integer valued random number that we used in the Needle in a Haystack protocol). After N rounds, process i outputs "attack" if and only if i has heard the value of *rfire* from process 1 and $count_i \geqslant rfire$.

Process i starts counting (i.e., sets $count_i$ to 1) in round r as soon it finds out that $(v_0, -1)$ and $(1, 0)$ flow to (i, r). These conditions, of course, are imposed because we are trying to calculate the modified level. To implement the first condition, we use a variable $valid_i$ at each process i that is set to *true* in the first round r such that $(v_0, -1)$ flows to (i, r). To implement the second condition, all processes other than process 1 initially set the value of $rfire_i$ to a special value *undefined* which is updated when a message is received with the value of *rfire*.

7.3. *Protocol Code*

Protocol S consists of local state machines, each of which has a set of states, an initial state, a state transition function, a message generation function, and an output decision function. We describe each component in turn:

Each process i has the following state variables:

- $count_i$: integer between 0 and N (counts the value of $ML_i^r(R)$ in the current run R).
- $rfire_i$: either a default value of *undefined* or a real number in the range $(0, 1/\varepsilon]$.
- $seen_i$: a subset of V (represents the processes that i knows have reached $count_i$).
- $valid_i$: a boolean (that is true if i has heard from v_0).

We also use three temporary variables at each process: $highcount_i$ (an integer), $highseen_i$ (a subset of V), and $highset_i$ (a set of messages, whose format we describe later.)

The initial states are as follows. Process 1 initially sets $rfire_1$ to a a random number uniformly distributed in the range $(0, 1/\varepsilon]$. All processes i other than 1, set $rfire_i =$ *undefined*. The $valid_i$ flag is set to be *true* if and only if process i has received an input message from v_0 in round 0. Process 1 initially sets $count_1 = 1$ and $seen_1 = \{1\}$ if process 1 has received an input message from v_0 in round 0; if not, process 1 initially sets $count_1 = 0$ and $seen_1 = \{\ \}$. All other processes i initially set $count_i = 0$ and $seen_i = \{\}$.

A message is denoted by m and has fields $m(rfire)$, $m(count)$, $m(seen)$, and $m(valid)$. The message generation function for i in every round sends a message $m(rfire, count, seen, valid)$ to all neighbors with $m(rfire) = rfire_i$, $m(count) = count_i$, $m(seen) = seen_i$, $m(valid) = valid_i$. Thus, i sends a message with its current state to all neighbors in every round.

At the end of a round r, for $1 \leqslant r \leqslant N$, process i executes the procedure PROCESS-MESSAGE(MR_i, i), where MR_i is the set of messages process i has received in round r. PROCESS-MESSAGE(MR_i, i) is shown in Fig. 5. The first four lines are used to decide when a process starts counting; the remainder of the code does the actual counting.

Finally, $O_i(q_i^N) = 1$ if and only if $rfire_i \neq undefined$ and $count_i \geqslant rfire_i$ at the end of N rounds.

7.4. *Proof of Properties of Protocol S*

Notation. Consider any execution $Ex(R, \alpha)$. Let v^r denote the value of a variable at time r (i.e., at the end of round r). For example, $count_i^r$ denotes the value of $count_i$ at time r. Define *rfire* to be the value of $rfire_1$ in the initial state.

Our first major step will be to establish that $count_i^r = ML_i^r(R)$. To allow a careful inductive proof, we will

PROCESS-MESSAGE(MR_i, i)

If ($rfire_i = undefined$) and ($\exists m \in MR_i : m(rfire) \neq undefined$)
    then $rfire_i := m(rfire)$
If ($valid_i = false$) and ($\exists m \in MR_i : m(valid) = true$)
    then $valid_i := true$
If ($valid_i = true$) and ($rfire_i \neq undefined$) and ($count_i = 0$) then
    $count_i := 1$
    $seen_i := \{i\}$;

If ($count_i \geq 1$) and ($MR_i \neq \emptyset$) then
    $highcount_i := Max_{m \in MR_i} m(count)$
    $highset_i := \{m \in MR_i : m(count) = highcount_i\}$
    $highseen_i := \cup_{m \in highset_i} m(seen)$
    If $highcount_i = count_i$ then
        $seen_i := seen_i \cup highseen_i \cup \{i\}$
    Else
        If $highcount_i > count_i$ then
            $seen_i := highseen_i \cup \{i\}$;
            $count_i := highcount_i$;
    If $seen_i = V$ then
        $count_i := count_i + 1$;
        $seen_i := \{i\}$;

FIG. 5 Procedure executed by process i at the end of a round in Protocol S.

introduce invariants. The invariants should be intuitively clear from the previous discussion; the proofs are deferred to the appendix.

LEMMA 7.3. *For any execution* $Ex(R, \alpha)$ *of Protocol S, the following assertions are true for all* r, $0 \leqslant r \leqslant N$, *and for all* $i, j \in V$:

1. $rfire_i^r$ *is either equal to rfire or undefined.*
2. $count_i^r \geqslant 1$ *if and only if* $rfire_i^r \neq undefined$ *and* $valid_i^r = true$.
3. $(1, 0)$ *flows to* (i, r) *if and only if* $rfire_i^r = rfire$.
4. $(v_0, -1)$ *flows to* (i, r) *if and only if* $valid_i^r = true$.
5. *If* (j, s) *flows to* (i, r) *in R, then either* $(count_i^r > count_j^s)$ *or* $(j \in seen_i^r$ *and* $count_i^r = count_j^s)$ *or* $(count_i^r = count_j^s = 0)$.
6. *If* $(j \in seen_i^r)$ *then there is some s such that* $(count_j^s = count_i^r)$ *and* (j, s) *flows to* (i, r) *in R.*
7. $seen_i^r \neq V$ *and* $seen_i^r \neq V - \{i\}$. *Also, if* $count_i^r \geqslant 1$ *then* $i \in seen_i^r$.
8. $ML_i^r \geqslant count_i^r$.

These invariants can now be used to establish that each process counts a value equal to its modified level measure. This should not be hard to believe since the code follows the definition of modified level.

LEMMA 7.4. *For all* $i \in V$, *any r such that* $0 \leqslant r \leqslant N$, *and any execution* $Ex(R, \alpha)$ *of Protocol S:* $count_i^r = ML_i^r(R)$.

Proof. From the last invariant in Lemma 7.3, we see that $count_i^r \leqslant ML_i^r(R)$. So we show that $count_i^r \geqslant ML_i^r(R)$. We do so by induction on the value of $ML_i^r(R)$.

First if $ML_i^r(R) = 0$ we are done trivially since $count_i^r$ is always nonnegative. We use $ML_i^r(R) = 1$ as the base case. Then from the definition of $ML_i^r(R)$, we know that $(v_0, -1)$ and $(1, 0)$ flow to (i, r) in run R. Hence by the third and fourth invariants in Lemma 7.3, $rfire_i^r = rfire$ and $valid_i^r = true$. Hence by the second invariant in Lemma 7.3, $count_i^r \geqslant 1$.

Next, suppose $ML_i^r(R) = l > 1$. Then from the definition of $ML_i^r(R)$, we know that for all $j \neq i$ there exists $r_j < r$ such that (j, r_j) flows to (i, r) in run R and $ML_j^{r_j} = l - 1$. By the induction hypothesis, $count_j^{r_j} = ML_j^{r_j} > 0$. Hence by the fifth invariant in Lemma 7.3 and the inductive hypothesis, either $count_i^r > l - 1$ (in which case we are done) or for all $j \neq i$, $j \in seen_i^r$. But the second case contradicts the seventh invariant in Lemma 7.3. ∎

Next we prove the validity, unsafety, and liveness properties of Protocol S.

THEOREM 7.5. (Protocol Validity). *Protocol S satisfies Validity.*

Proof. Informally, in any execution in which no process receives an input signal, no process hears from v_0, and so $count_i^N = 0$ for all i. Thus, by the output decision function, $O_i(q_i^N) = 0$ for all i in this execution.

More formally, fix a run R such that $I(R) = \{\ \}$, a random vector α, and any process i. Consider the execution $Ex(R, \alpha)$. Thus $(v_0, -1)$ does not flow to (i, N) for any $i \in V$. Thus by invariant 4 in Lemma 7.3, $valid_i^N = false$. Hence by Lemma 7.3, invariant 2, $count_i^N = 0$. Next, by invariant 1, Lemma 7.3, $rfire_i^N$ is either equal to $rfire$ (which is strictly greater than 0) or *undefined*. In either case, by the output decision function, $O_i(q_i^N) = 0$ in $Ex(R, \alpha)$. ∎

To prove the unsafety and liveness properties of S we characterize when the total attack and no attack events occur. Let *Mincount* be the minimum across all processes i of the value of $count_i$ at the end of an execution. The next lemma states that all generals will attack if *Mincount* is no less than *rfire*, and no general will attack if *Mincount* is strictly less than $rfire - 1$.

LEMMA 7.6. *Fix an execution E of Protocol S. If* $Mincount \geqslant rfire$ *then* $E \in TA$. *If* $Mincount < rfire - 1$, *then* $E \in NA$.

Proof. If $Mincount \geqslant rfire$ then for all processes i, $count_i^N \geqslant rfire$. But $rfire > 0$. Thus for all i, $count_i^N \geqslant 1$. It follows (by Lemma 7.3, invariants 1 and 2) that for all i, $rfire_i^N = rfire$. Hence for all i, $count_i^N \geqslant rfire_i^N$ and $rfire_i^N \neq undefined$. Thus for all i, (using the decision function), $O_i(q_i^N) = 1$. Hence, $E \in TA$.

If $Mincount < rfire - 1$, then using Lemma 7.4 and using the fact that the modified level measured at any two processes differs by at most 1 (Lemma 7.2), for all i, $count_i^N < rfire$. Now (by Lemma 7.3, invariant 1), either $rfire_i^N = rfire$ or $rfire_i^N = undefined$. Hence, for all $i \in V$, either $count_i^N < rfire_i^N$ or $rfire_i^N = undefined$. Thus by the definition of the output decision function, $O_i(q_i^N) = 0$ for all i. Hence $E \in NA$. ∎

THEOREM (Protocol Unsafety). *S satisfies Agreement with parameter* ε.

Proof. By definition $U_s(S)$ is the maximum across all runs R of $Pr[PA, R]$. Consider any execution $E = Ex(R, \alpha)$. Now partial attack PA is the complement of the no attack and total attack events, NA and TA. From Lemma 7.6, we know that either TA or NA will occur unless $Mincount < rfire \leqslant Mincount + 1$. Hence $Pr[PA, R] \leqslant Pr[Mincount < rfire \leqslant Mincount + 1, R]$. Now for a given R, *Mincount* is fixed while *rfire* is a uniformly distributed random number in the range $(0, 1/\varepsilon]$. Thus $U_s(s) \leqslant \varepsilon$. ∎

THEOREM (Protocol Liveness). $\mathscr{L}(S, R) \geqslant Min(1, \varepsilon ML(R))$.

Proof. Recall the definition of $\mathscr{L}(S, R)$ as the probability of total attack, $Pr[TA, R]$. We find a lower bound on

$Pr[TA, R]$. Consider any execution E. From Lemma 7.6, $E \in TA$ if $Mincount \geqslant rfire$. But by Lemma 7.4 and the definition of $Mincount$, $Mincount = ML(R)$. Hence, $E \in TA$ if $ML(R) \geqslant rfire$. Thus for any run R, $Pr[TA, R]$ is no less than $Pr[ML(R) \geqslant rfire, R]$. Now for a given R, $ML(R)$ is fixed while $rfire$ is a uniformly distributed random number in the range $(0, 1/\varepsilon]$. Thus $Pr[TA, R]$ is no less than $Min(1, \varepsilon ML(R))$. ∎

8. CLOSING THE GAP: A SECOND LOWER BOUND

Theorem 6.4 states that for every run R and every protocol F, the liveness $\mathscr{L}(F, R)$ of any protocol F is at most $Min(1, \varepsilon L(R))$. We described a protocol S whose liveness is $Min(1, \varepsilon ML(R))$. From Lemma 7.1, we know that $ML(R)$ differs from $L(R)$ by at most one. Thus we have a small but irritating gap of ε. Our second lower bound shows, under a reasonable set of conditions that we call the usual case assumption, that no protocol F can do better than $\varepsilon ML(R)$ on all runs R. More precisely, if any protocol F has a run R such that $\mathscr{L}(F, R) > \varepsilon ML(R)$ then there is some other run $\tilde{R}$ such that $\mathscr{L}(F, \tilde{R}) < \varepsilon ML(\tilde{R})$. Thus together the two bounds show that Protocol S is indeed "optimal."

We note that the proof of the first lower bound is similar to the chain arguments used often in deterministic impossibility results [6]. However, in proving the second lower bound, we show a simple connection between causality and probabilistic agreement that may be interesting in its own right. This is explored in the definition of *causal independence*, and Lemmas 8.2 and 8.3.

The second lower bound needs the following assumption. We say that the *usual case assumption* holds if:

- The graph G is connected and the diameter of G is no more than the number of rounds N.
- $\varepsilon < 0.5$.

It is easy to see that these two conditions capture the usual and interesting cases. If the first condition does not hold then it can be shown that $L_i(R) \leqslant 1$ for all i, R, F and so by Lemma 6.4, $\mathscr{L}(F, R) \leqslant \varepsilon$. Similarly if the second condition does not hold, the protocol is allowed to fail more than half of the time. Thus the conditions preclude parameter settings that force absurdly small values of liveness and allow absurdly large values of unsafety.

THEOREM 8.1. (Refined Lower Bound). *Under the usual case assumption, if any protocol F has a run R such that $\mathscr{L}(F, R) > \varepsilon ML(R)$ then there is some other run $\tilde{R}$ such that $\mathscr{L}(F, \tilde{R}) < \varepsilon ML(\tilde{R})$.*

The proof exploits a simple connection between probabilistic independence and what we call causal independence. For any $i, j \in V$, we say that i and j are *causally independent* in run R if there is no $k \in V$ such that $(k, 0)$ flows to (i, N) and $(k, 0)$ flows to (j, N) in R. The connection between probabilistic independence and causal independence is expressed by the intuitive lemma:

LEMMA 8.2. *If i and j are causally independent in run R then the events (D_i, R) and (D_j, R) are independent events.*

Proof. Let $Y_k(R) = \{l: (l, 0)$ flows to (k, N) in $R\}$. Thus, if i and j are causally independent in R, then $Y_i(R) \cap Y_j(R) = \varnothing$. Thus, for any execution that follows run R, the decision events at i and j are based on information (and random bits) received from disjoint sets of processes. This implies the independence of the two events. ∎

If i and j are causally independent in run R, then there must be some restrictions on their decision probabilities in R in order to preserve the Agreement property. There are several ways in which these restriction can be phrased; we select one that is sufficient for the later development.

LEMMA 8.3. *Consider a run R in which i and j are causally independent and such that $Pr[D_i, R] = \varepsilon$. Then if $\varepsilon < 0.5$, $Pr[D_j, R] = 0$.*

Proof. Let $Pr[D_j, R] = \delta$. We know that $Pr[PA, R] \geqslant Pr[D_i \overline{D_j}, R] + Pr[D_j \overline{D_i}, R]$. But since i and j are causally independent in R we have by Lemma 8.2 that the events (D_i, R) and (D_j, R) are independent. Hence, $Pr[PA, R] \geqslant \varepsilon(1-\delta) + \delta(1-\varepsilon)$ and so $Pr[PA, R] \geqslant \varepsilon + \delta(1-2\varepsilon)$. But since $\varepsilon < 0.5$, $1 - 2\varepsilon > 0$. Hence by Agreement, $\delta = 0$. ∎

For the next lemma, recall the definition of $ML_i(R)$, the modified level of process i in run R. This lemma serves to set up the proof of the following lemma, Lemma 8.5.

LEMMA 8.4. *Suppose that for all runs R and for all $i \in V$, $Pr[D_i, R] = 0$ if $ML_i(R) = 0$. Then for all R and $i \in V$, $Pr[D_i, R] \leqslant ML_i(R)\varepsilon$.*

Proof. By induction on the value of $ML_i(R)$. Let $ML_i(R) = l$.

Base Case, $l = 0$: This is the assumption of the lemma.

Inductive Step, $l > 0$: Using a lemma similar to Lemma 6.2 we can show that if $\tilde{R} = Clip_i(R)$, then there is some k such that $ML_k(\tilde{R}) = l - 1$. Hence by inductive assumption, $Pr[D_k, \tilde{R}] \leqslant \varepsilon(l-1)$. Hence by Lemma 2.2,, $Pr[D_i, \tilde{R}] \leqslant \varepsilon l$. But by Lemma 5.2 and Lemma 2.1, $Pr[D_i, R] = Pr[D_i, \tilde{R}] \leqslant \varepsilon l$. ∎

Now consider a run R_1 in which only process 1 receives an input message and no other message is delivered in the run. The next lemma states that if the probability of process 1 outputting attack in this run is exactly ε, then we can prove a tighter lower bound on the decision probabilities than the bound of Lemma 6.3. Recall that the bound in Lemma 6.3 was stated in terms of $L_i(R)$.

LEMMA 8.5. *Suppose that* $R_1 = \{(v_0, 1, 0)\}$, $Pr[D_1, R_1] = \varepsilon$ *and* $\varepsilon < 0.5$. *Then for all runs* R *and all* $i \in V$, $Pr[D_i, R] \leqslant ML_i(R)\varepsilon$.

Proof. Consider any i and any R such that $ML_i(R) = 0$. Then we will claim that $Pr[D_i, R] = 0$. To do this we consider two cases, one of which must be true if $ML_i(R) = 0$.

- $(v_0, -1)$ does not flow to (i, N) in R. Then $L_i(R) = 0$ and hence by Lemma 6.3, $Pr[D_i, R] = 0$.
- $(1, 0)$ does not flow to (i, N) in R. Thus, $i \neq 1$ as $(1, 0)$ flows to $(1, N)$. Consider the run $Clip_i(R)$. From the definition of clipping, there is no tuple $(*, 1, *)$ in $Clip_i(R)$, because if there was, $(1, 0)$ would flow to (i, N) in R. Consider the run $\tilde{R} = Clip_i(R) \cup \{(v_0, 1, 0)\}$. By construction, the only tuple of the form $(*, 1, *)$ in $\tilde{R}$ is $(v_0, 1, 0)$. Hence 1 and i are causally independent in $\tilde{R}$.

  Also, $R_1 = Clip_1(\tilde{R})$ and hence $R_1 \overset{1}{\equiv} \tilde{R}$. Thus, $Pr[D_1, \tilde{R}] = \varepsilon$. Hence by Lemma 8.3, $Pr[D_i, \tilde{R}] = 0$. But $Clip_i(\tilde{R}) = Clip_i(R)$ and so by Lemma 5.2 and Lemma 2.1, $Pr[D_i, R] = Pr[D_i, \tilde{R}] = 0$.

Thus, in either case, we have shown that for any i and R, $Pr[D_i, R] = 0$ if $ML_i(R) = 0$. The lemma now follows from Lemma 8.4. ∎

LEMMA 8.6. *Suppose the graph* G *is connected and has diameter no more than* N. *Then there is a run* R *such that* $ML_1(R) = ML(R) = 1$, *and the only tuple of the form* $(*, 1, *)$ *is* $(v_0, 1, 0)$.

Proof. Let T be a spanning tree of G with 1 as the root. Such a tree exists because G is connected. Next we define R as follows.

- $I(R) = \{(v_0, 1, 0)\}$ (i.e., only process 1 receives input).
- For all $i, j \in V$ and $1 \leqslant r \leqslant N$, $(i, j, r) \in R$ if and only if i is the parent of j in the tree. (i.e., information only flows down the tree.)

It is not hard to see that since the height of the tree is no more than N, $ML_1(R) = 1$ and $ML_i(R) \geqslant 1$ for all $i \in V$. Thus, $ML(R) = 1$. ∎

We now return to the proof of Theorem 8.1.

Proof. Suppose there is some protocol F such that for all R, $\mathscr{L}(F, R) \geqslant \varepsilon ML(R)$.

By Lemma 8.6, there is a run R_1 such that $ML_1(R_1) = ML(R_1) = 1$ and the only tuple of the form $(*, 1, *)$ in R_1 is $(v_0, 1, 0)$. It is easy to verify that $L_1(R_1) = 1$.

Thus, by assumption, $\mathscr{L}(S, R_1) \geqslant \varepsilon ML(R_1) = \varepsilon$. Thus, by Lemma 2.3, $Pr[D_1, R_1] \geqslant \varepsilon$. Also, by Lemma 6.3, since $L_1(R_1) = 1$, $Pr[D_1, R_1] \leqslant \varepsilon$. Hence, $Pr[D_1, R_1] = \varepsilon$.

Now consider the run $R_2 = Clip_1(R_1) = \{(v_0, 1, 0)\}$. Then by Lemma 5.2, $R_2 \overset{1}{\equiv} R_1$. Thus, by Lemma 2.1, $Pr[D_1, R_2] = \varepsilon$. Hence by Lemma 8.5, for all i, R, $Pr[D_i, R] \leqslant \varepsilon ML_i(R)$. Thus, for all R, $Min_i Pr[D_i, R] \leqslant Min_i\, \varepsilon ML_i(R)$. Thus, from Lemma 2.3 and the definition of $ML(R)$, $\mathscr{L}(F, R) \leqslant \varepsilon ML(R)$.

Thus, we have shown that for any protocol F, if for all R, $\mathscr{L}(F, R) \geqslant \varepsilon ML(R)$, then for all R, $\mathscr{L}(F, R) = \varepsilon ML(R)$. This implies the theorem. ∎

9. CONCLUSIONS

A solution to coordinated attack is important in situations where consensus must be reached across unreliable links and within a specified time constraint. This can arise in real-time applications. The lack of a good solution to coordinated attack has also lead network protocol designers to accept as inevitable the possibility of inconsistent intermediate states during transition periods. This in turn can lead to other effects, such as packet looping, when routes change [17].

Thus an effective solution to the coordinated attack problem could be quite useful for many real problems. Since it is well known that no deterministic solution exists, this paper investigates whether randomized solutions can do better, for a practical model in which link crashes can result in arbitrary message loss. Our results, however, are pessimistic.

For coordinated attack against a strong adversary, we have shown that no protocol can achieve a tradeoff between liveness and safety ($\mathscr{L}/U$) that is better than linear in the number of rounds. This is bad news. For example if we want to achieve liveness with probability 1 on some run, and yet limit the probability of error to be less than 0.001, then the protocol must run for at least 1000 rounds. Protocol S demonstrates that the lower bounds are tight, but its performance is far from adequate.

In practice, there are two approaches that may help us to overcome these limitations. One approach is to add redundant links and assume that failures can only affect some fraction of the links in the network; then simple majority voting techniques can be used. However, this approach is expensive. The other approach is to assume a weaker failure model than a strong adversary. One such adversary, which we call a *weak adversary*, is a *probabilistic* adversary which can destroy messages with a probability p that is not known in advance. We have preliminary proof sketches of results that show vastly improved performance against such an adversary. However, it is not at all clear that such a weak adversary is realistic.

While our results are stated in a synchronous model, it seems clear that they can be extended to an asynchronous model. The concepts of causal clipping and information level carry over to an asynchronous setting. Our "optimal" protocol can be modified to work in an asynchronous setting. Most real networks, however, are best modeled by

a semi-synchronous model such as [14]; in this model the delay on network links is variable but has an upper bound. In such a setting, the lower bound would translate into a limitation on the achievable liveness given that the protocol is given a fixed amount of time to execute. However, careful proofs of these results are needed to confirm this intuition.

Since an adversary is a set of runs, we have limited ourselves to a deterministic, oblivious adversary whose behavior is not adaptive. For example, our "strong" adversary cannot use knowledge of message contents to control message delivery. However, our lower bounds are pessimistic. Thus, there does not seem to be any point in considering stronger adversaries.

Finally, in terms of methodology, we were pleased that the lower bound led us almost immediately to the optimal protocol. The optimal protocol calculates (and exploits) a small modification of the level measure that arises in the statement of the first lower bound.

APPENDIX A. PROOF OF INVARIANTS

We begin with a simple definition and some useful facts.

DEFINITION A.1. A property P is monotonic in any execution E of Protocol S if whenever P is true at some round r of E then P is true at all rounds greater than r.

LEMMA A.1. (Monotonicity). *The following properties are monotonic in any execution E of Protocol S*:

- $count_i^r \geqslant l$ *for any* l.
- $valid_i^r = true$.
- $rfire_i^r = rfire$.
- $rfire_i^r \neq undefined$.

Proof. Immediate from examination of code. ∎

We now return to the proof of the invariants. For convenience we repeat the lemma and then provide the proof.

LEMMA 7.3. *For any execution* $Ex(R, \alpha)$ *of Protocol S, the following assertions are true for* $0 \leqslant r \leqslant N$ *and for all* $i, j \in V$:

1. $rfire_i^r$ *is either equal to rfire or undefined.*

2. $count_i^r \geqslant 1$ *if and only if* $rfire_i^r \neq undefined$ *and* $valid_i^r = true$.

3. $(1, 0)$ *flows to* (i, r) *if and only if* $rfire_i^r = rfire$.

4. $(v_0, -1)$ *flows to* (i, r) *if and only if* $valid_i^r = true$.

5. • *If* (j, s) *flows to* (i, r) *in R then either* $(count_i^r > count_j^s)$ *or* $(j \in seen_i^r$ *and* $count_i^r = count_j^s)$ *or* $(count_i^r = count_j^s = 0)$.

• *If* $(j \in seen_i^r)$ *then there is some s such that* $(count_j^s = count_i^r)$ *and* (j, s) *flows to* (i, r) *in R.*

6. $seen_i^r \neq V$ *and* $seen_i^r \neq V - \{i\}$. *If* $count_i^r > 0$ *then* $i \in seen_i^r$.

7. $ML_i^r \geqslant count_i^r$.

Proof. The proof is by induction on r. The base case $r = 0$ is easily seen to be satisfied by the initial states s_i^0 and s_i^1. For the inductive step, we assume that all the invariants are true for rounds less than r for $r > 0$. We prove each invariant in turn:

1. We know from the code that $rfire_i^r$ is different from $rfire_i^{r-1}$ if and only if i receives a message m in round r from some process j and $rfire_i^{r-1} = undefined$. But in that case $rfire_i^r = m(rfire) = rfire_j^{r-1}$. Then we are done because $rfire_j^{r-1}$ is either *rfire* or *undefined* by the inductive hypothesis.

2. Let G_i^r denote $valid_i^r = true$ and $rfire_i^r \neq undefined$. From the code, if $count_i^{r-1} = 0$ and $count_i^r \geqslant 1$, then $G_i^r = true$. Similarly, from the code it is easy to see that if $G_i^{r-1} = false$ and $G_i^r = true$, then $count_i^r \geqslant 1$. Also by monotonicity (Lemma A.1), if $G_i^{r-1} = true$ then $G_i^r = true$. Similarly by monotonicity (Lemma A.1), if $count_i^{r-1} \geqslant 1$ then $count_i^r \geqslant 1$.

3. First we prove that if $(1, 0)$ flows to (i, r) then $rfire_i^r = rfire$. If $(1, 0)$ flows to $(i, r-1)$ then by inductive hypothesis, $rfire_i^{r-1} = rfire$. But $(i, r-1)$ flows to (i, r) and hence $(1, 0)$ flows to (i, r) by transitivity (Lemma 5.1). Also by monotonicity (Lemma A.1), $rfire_i^r = rfire$. On the other hand, if $(1, 0)$ does not flow to $(i, r-1)$ and $(1, 0)$ flows to (i, r) then there must be some j such that $(j, i, r) \in R$ and $(1, 0)$ flows to $(j, r-1)$. Hence by the inductive hypothesis, $rfire_j^{r-1} = rfire$. Thus, if m is the message sent by j to i in round r, then $m(rfire) = rfire$. Hence by the code $rfire_i^r = rfire$.

Next, we prove that if $rfire_i^r = rfire$ then $(1, 0)$ flows to (i, r). If $rfire_i^{r-1} = rfire$ then $(1, 0)$ flows to $(i, r-1)$ by inductive hypothesis and by transitivity (Lemma 5.1), $(1, 0)$ flows to (i, r). Also by Monotonicity (Lemma A.1), $rfire_i^{r-1} = rfire_i^r$, and so we are done. On the other hand if $rfire_i^{r-1} \neq rfire$ but $rfire_i^r = rfire$ then there must be some $m \in MR_i^r$, sent by say k, such that $m(rfire) = rfire$. Thus, $(k, i, r) \in R$ and $rfire_k^{r-1} = rfire$. Hence we are done by inductive hypothesis and transitivity (Lemma 5.1).

4. The proof of this invariant is identical to the previous invariant except that we replace $(1, 0)$ by $(v_0, -1)$ and $(rfire_i^r = rfire)$ by $(valid_i^r = true)$.

5. We prove each part in turn:

• If (j, s) flows to $(i, r-1)$ then by inductive hypothesis, either $(count_i^{r-1} > count_j^s)$ or $(j \in seen_i^{r-1}$ and

$count_i^{r-1} = count_j^s$) or ($count_i^{r-1} = count_j^s = 0$). But $(i, r-1)$ flows to (i, r) and hence (j, s) flows to (i, r) by transitivity (Lemma 5.1). Also by monotonicity (Lemma A.1), if $count_i^{r-1} > count_j^s$, then $count_i^r > count_j^s$. Also if $j \in seen_i^{r-1}$ but $j \notin seen_i^r$, then we see from the code that $count_i^r > count_i^{r-1}$.

If (j, s) does not flow to $(i, r-1)$ and $j \neq i$ and (j, s) flows to (i, r) then there must be some k such that $(k, i, r) \in R$ and (j, s) flows to $(k, r-1)$. Hence by the inductive hypothesis, either ($count_k^{r-1} > count_j^s$) or ($j \in seen_k^{r-1}$ and $count_k^{r-1} = count_j^s$) or ($count_k^{r-1} = count_j^s = 0$). But in round r, i receives a message m from k with $m(count) = count_k^{r-1}$ and $m(seen) = seen_k^{r-1}$. It is not hard to see from the code that in either of the three cases above, at time r the invariant is preserved.

- If $j \in seen_i^r$ then we have two possibilities. If $j \in seen_i^{r-1}$ and $count_i^r = count_i^{r-1}$ then we are done by inductive hypothesis and transitivity (Lemma 5.1). If $j \notin seen_i^{r-1}$ or $count_i^r > count_i^{r-1}$ then there must be some $m \in MR_i^r$, sent by say k, such that $m(count) = count_i^r$ and $j \in m(seen)$. Hence $j \in seen_k^{r-1}$ and we are done by inductive hypothesis and transitivity (Lemma 5.1).

6. Consider first proving that $seen_i^r \neq V$ and $seen_i^r \neq V - \{i\}$. If $seen_i^r = seen_i^{r-1}$ we are done by the inductive hypothesis. So suppose $seen_i^r \neq seen_i^{r-1}$. Then there must be some message m received by i in round r from say j such that $seen_i$ changes after processing m. But by examining the code, we see that the desired property is preserved after processing m. In particular, after processing m, if $seen_i$ changes, then i is added to $seen_i$. Also if $seen_i = V$ during the processing of m, then $seen_i$ is reset to $\{i\}$. Recall that $|V| > 1$.

Now consider proving that if $count_i^r > 0$, then $i \in seen_i^r$. Observe that whenever the code changes $count_i$, it adds i to $seen_i$.

7. If $count_i^r = 0$ we are done trivially. If $count_i^r = 1$, by invariants 1 and 2, $rfire_i^r = rfire$ and $valid_i^r = true$. Thus, by invariants 3 and 4, $(1, 0)$ and $(v_0, -1)$ flow to (i, r) in R. Hence $ML_i^r \geqslant 1$.

If $count_i^r > 1$ and $count_i^r = count_i^{r-1}$ we are done by inductive hypothesis since $count_i^r \leqslant ML_i^{r-1} \leqslant ML_i^r$. If $count_i^r > 1$ and $count_i^r \neq count_i^{r-1}$ then we have two more possibilities.

- If $highcount_i^r = count_i^r$ then there must be some message m, sent by say j, with $m(count) = count_i^r$ and $count_i^r = count_j^{r-1}$. Thus, by the inductive hypothesis, $count_i^r \leqslant ML_j^{r-1}$. Thus, $ML_j^{r-1} > 1$ and so for all $k \neq j$ there must be some r_k such that (k, r_k) flows to $(j, r-1)$ and $ML_k^{r_k} = ML_j^{r-1} - 1$. By transitivity (Lemma 5.1), (k, r_k) flows to (i, r). Also, $(j, r-1)$ flows to (i, r). Thus, for all $k \neq i$ there must be some r_k such that (k, r_k) flows to (i, r) and $ML_j^{r_k} \geqslant ML_j^{r-1} - 1$. Hence since $ML_j^{r-1} > 1$, $ML_i^r \geqslant ML_j^{r-1}$. Thus, $count_i^r \leqslant ML_i^r$.

- If $highcount_i^r + 1 = count_i^r$ then for all $k \neq i$ either:

  — There must be some message m, sent by say j to i in round r, with $m(count) = highcount_i^r$ and $k \in m(seen)$ OR

  — $highcount_i^r = count_i^{r-1}$ and $k \in seen_i^{r-1}$.

In either case there is some l such that $k \in seen_l^{r-1}$ and $count_l^{r-1} = count_i^r - 1$ and such that $(l, r-1)$ flows to (i, r). Thus, by invariant 5, and Transitivity (Lemma 5.1), there is some s such that (k, s) flows to (i, r) and $count_k^s = count_i^r - 1$. Also, by the inductive hypothesis, $ML_k^s \geqslant count_k^s$. Since this is true for all $k \neq i$, it is clear from the definition that i can reach m-height $count_i^r$. Thus, $ML_i^r \geqslant count_i^r$. ∎

ACKNOWLEDGMENTS

We are grateful to Hagit Attiya, Baruch Awerbuch, Mihir Bellare, Cynthia Dwork, Ken Goldman, Gil Neiger, Steve Ponzio, and Larry Stockmeyer for their help and suggestions.

Received March 31, 1995; final manuscript received April 1, 1996

REFERENCES

1. M. Ben-Or, Another advantage of free choice, *in* "Proceedings, 2nd ACM Symposium on Principles of Distributed Computing, 1983," pp. 27–30.
2. P. Bernstein, V. Hadzalicos, and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison–Wesley, Reading, MA, 1986.
3. B. Chor and C. Dwork, Randomization in Byzantine agreement, *Adv. Comput. Res.* **5**, 443–497, (1989).
4. K. M. Chandy and J. Misra, How processes learn, *Distrib. Comput.* **1**(1), 40–52 (1986).
5. D. Dolev, The Byzantine generals strike again. *J. Algor.* **3**, 14–30 (1982).
6. M. Fischer and N. Lynch, A lower bound for the time to assure interactive consistency, *Inform. Process. Lett.* **14**(4), 183-186, (1982).
7. R. Graham and A. Yao, On the improbability of reaching Byzantine agreements. *in* "Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, Seattle, WA, 1989," pp. 467–478.
8. J. Gray, "Notes on Data Base Operating Systems," Technical Report, IBM Report RJ2183(30001), IBM, February 1978, 1989, pp. 291–302; also *in* Operating Systems: An Advanced Course," Lecture Notes in Computer Science, No. 60, Springer-Verlag, Berlin/New York.
9. V. Hadzilacos, Connectivity requirements for Byzantine agreement under restricted Types of failures, *Distrib. Comput.* **2**(2), 95–103 (1987).
10. J. Halpern and Y. Moses, Knowledge and common knowledge in a distributed environment, *in* "Proceedings of the 3rd Annual Symposium on Principles of Distributed Computing, 1984," pp. 50–61.
11. A. Karlin and A. Yao, Probabilistic lower bounds for Byzantine agreement, unpublished manuscript.
12. L. Lamport, Time clocks and the ordering of events in a distributed system, *Comm. ACM* **21**(7), 558–565 (1977).
13. L. Lamport, R. Shostak, and M. Pease, The Byzantine generals problem, *ACM Trans. Programming Languages Systems* **4**(3), 382–401 (1981).

14. M. Merritt, F. Modugno, and M. Tuttle, Time constrained automata, *in* "CONCUR 91," pp. 408-423.
15. Y. Moses and M. Tuttle, Programming simultaneous actions using common knowledge, *Algorithmica* **3**(1), 121–169 (1988).
16. M. Rabin and D. Lehmann, On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem, *in* "Proceedings of 8th ACM Symposium on Principles of Programming Languages, 1981," pp. 133–138.
17. A. Tanenbaum, "Computer networks," 2nd ed., Prentice–Hall, New York, 1989.
18. Y. Moses and M. Tuttle, Programming simultaneous actions using common knowledge, *Algorithmica* **3**(1), 121–169 (1988).

Printed in Belgium